

Feature Model Configuration Reuse Scheme for Self-Adaptive Systems

Sumaya Alkubaisi^{1,*}, Said Ghoul² and Oguz Ata¹

¹Electrical and Computer Engineering Department, Altinbas University, Istanbul, 34218, Turkey

²Faculty of Information Technology, Research Laboratory on Bio-Inspired Software Engineering, Philadelphia University, Amman, 26160, Jordan

*Corresponding Author: Sumaya Alkubaisi. Email: sumayah.al-kubaisi@ogr.altinbas.edu.tr
Received: 13 April 2021; Accepted: 21 May 2021

Abstract: Most large-scale systems including self-adaptive systems utilize feature models (FMs) to represent their complex architectures and benefit from the reuse of commonalities and variability information. Self-adaptive systems (SASs) are capable of reconfiguring themselves during the run time to satisfy the scenarios of the requisite contexts. However, reconfiguration of SASs corresponding to each adaptation of the system requires significant computational time and resources. The process of configuration reuse can be a better alternative to some contexts to reduce computational time, effort and error-prone. Nevertheless, systems' complexity can be reduced while the development process of systems by reusing elements or components. FMs are considered one of the new ways of reuse process that are able to introduce new opportunities for the reuse process beyond the conventional system components. While current FM-based modelling techniques represent, manage, and reuse elementary features to model SASs concepts, modeling and reusing configurations have not yet been considered. In this context, this study presents an extension to FMs by introducing and managing configuration features and their reuse process. Evaluation results demonstrate that reusing configuration features reduces the effort and time required by a reconfiguration process during the run time to meet the required scenario according to the current context.

Keywords: Self-adaptive system; feature model; system reuse; configuration management; variability modeling

1 Introduction

Large systems have become progressively more complicated over the recent years, leading to increased focus on system maintenance and configurations. Self-adaptive systems (SASs) are considered to be of great importance in the handling of complex systems [1]. They are capable of reconfiguring themselves during the run time to satisfy the scenarios of the requisite contexts (corresponding to changes in user requirements, systems themselves or their environments). However, reconfiguration process need to be triggered every time the system is adapted [2]. In this context, configuration reuse is a prospective alternative to reduce time, effort and error-prone. The



This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

reuse of software involves the reuse of appropriate available software components to reduce cost or time, with slight or no modifications to the components [3]. Complexity can be decreased in the development of systems by reusing components or elements. FMs are one of the new forms of reuse that are able to bring new opportunities for the reuse process beyond the conventional software components [4]. Reusability has previously been applied to SASs for example: Rainbow strategy, MADAM framework, MUSIC framework, and ENTRUST strategy [5–8] respectively. However, none has produced a design or developed a self-adaptive system with reuse process. Moreover, the formal modeling of a configuration reuse process or reused configurations remains lacking, especially with respect to system feature models (FMs).

Several large-scale systems, including SASs, use FMs to represent their complex architectures [9], by managing, expressing, and reusing commonalities and variability information among their products. Each feature represents a distinct ability of the system. The reuse of features reduces time, cost and effort and improves the quality of the product as well as that of the software process [10–12].

Several studies have been conducted on FMs to address the variability and complexity of constituent systems, and preserve their alignment to variable systems [13–17]. Certain works have focused on the management and improvement of FM customization, which is a popular method to reduce cost and improve user satisfaction [18–21]. However, the software configuration is yet to be modeled as a feature in the FMs. Other studies [22–25] have incorporated FMs into systems adaptation process, but no notions or extensions have been introduced into FMs to enable them to model these adaptation processes or their reused processes. A few works have enriched FMs using concepts and extensions to enable them to model SASs. The appended concepts, including elementary features, equipped with their relations, processes and constraints, are designed to ensure coherence [2,26,27]. All of the aforementioned works have focused on modeling and reusing elementary features without considering the management of configuration reuse, except for [27], in which the authors modeled the concept of metamorphosis, which involves the grouping of features (configuration). To this end, they modeled the configuration using an FM, but did not model the reuse process. In one of the most significant works in the domain [2], the authors introduced a framework for software product lines (SPLs), enabling the modeling of contextual influences, user customization and evolution. In particular, a hybrid feature model (HyFM) was presented. It supports the implementation of concepts such as contextual information, validity formula, cross-tree constraint and version awareness. However, this approach was not designed to support management of configuration reuse. Moreover, a reconfiguration process is required to be triggered every time a change in the context occurs or an artifact of the system is evolved by an engineer. This introduces additional overhead in the form of significant time and resources. To address the aforementioned lacuna, this study proposes an extension to HyFM2 to handle configuration features management that supports its modeling and coherent reuse. This extension includes new features (configurations), context information and constraints to ensure coherency among features. Configuration feature management process is defined in order to manage configuration features and their reuse process from creation to destruction. The remainder of this paper is structured as follows. A brief background on FMs on which our work is founded is described in Section 2. An illustrative example and the enhanced feature model (EnFM) are presented in Section 3. The modeling of configuration features using the FM is presented in Section 4. A case study is detailed in Section 5. The method proposed in this paper is evaluated in Section 6. Finally, a conclusion and future directions of research are presented in Section 7.

2 Background

2.1 Feature Models (FMs)

FMs are hierarchal representations of a system's commonalities and variabilities in terms of features. Each feature in the FM represents a distinctive ability or functionality of the system [2]. The top feature of an FM is called root or concept. Each feature can possess multiple child features. Features are either mandatory or optional, and arranged alternatively (only one feature is selected) or in groups (at least one feature is selected). A feature attribute is a typed variable used to define additional characteristics of a feature [28]. To capture the different states of evolution of features, feature versions are included in the definition of FMs — a user might prefer an older implementation of features over a newer implementation to maintain compatibility with other systems [29]. Further, cross-tree constraints (CTCs) [2] are Boolean formulas that are used to define additional dependencies among the features that are not defined by the structure of the FMs, feature attributes' expressions, or range of a feature's version. Contextual information [2] captures the environmental situation and external parameters, and is represented using identifier-value pairs. Each context is assigned a value, and each value is connected to certain features in the FM via the validity formula (VF), which is a propositional formula that enforces the selection of a certain feature or attribute depending on the context. FMs also include the notion of user profiles [30] to allow the transformation of one configuration into another during run time in order to accommodate multiple users who might require different configurations of the same system under varying contexts.

2.2 Configuration Features

Configuration features comprise coherent features of a system along with a sorted collection of their relations and constraints to specify a given product in an SPL. Configuration features might be reused under certain system conditions. Such reuse avoids the necessity to reconfigure the system for each change in the environment, consequently reducing cost and complexity.

3 Illustrative Example

In this section, an example of a car system, which is a dynamic software product line (DSPL), obtained from [2], is presented to illustrate the concepts introduced in this paper. Fig. 1 [2] depicts the feature model of a car system comprising two electronic control units (ECUs)—a mandatory ECU_A and an optional ECU_B. ECU_A is responsible for the emergency call system of the car—it triggers a preconfigured call with the satellite position of the car and additional relevant data when an emergency is detected. The car considered in this study is equipped with two emergency call systems—one for Russia called ERAGlonass and another for Europe called eCall. To maximize fuel and performance efficiency, the GearAdvice feature, offered by the optional ECU_B, suggests a suitable gear to use based on three options for driving style—family, natural, or sports. Four types of context information are captured using independent sensors—pollution represents the number of contaminants in the air, Location captures the current position of the car, GearViolation captures the number suboptimal uses of the gearshift by the driver, and FuelPercentage captures the remaining fuel level.

4 Modeling of Configuration Features

4.1 Enhanced FM with Configuration – Example

The example presented in Fig. 2 presents a novel FM and the updates introduced to the FM, as depicted in Fig. 1, by the introduction of the new configuration feature. This example (Fig. 2)

will be used in this section to illustrate the introduced concepts that support management of configuration features.

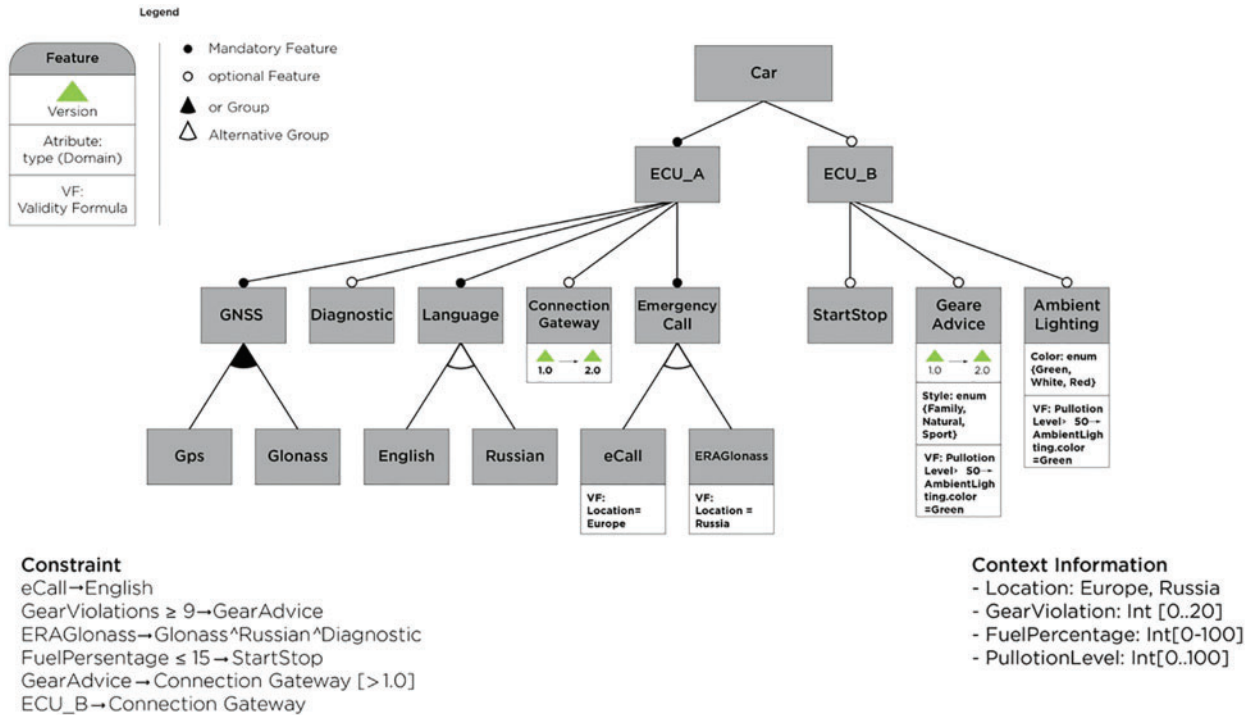


Figure 1: Feature model for a car system [2]

4.2 Definitions

The introduction of configuration features necessitates the definition of the following new concepts (depicted in Fig. 2):

- A ConfigReuse sensor that captures the number of times a reconfiguration process is triggered to achieve the same configuration. When the same configuration is created more than 2 times, it is saved for future reuse.
- A ConfigAvailable sensor that captures the availability of a configuration. When a configuration is saved, the value of this sensor is set to be true.
- A Granularity attribute that fixes the feature type (elementary or configuration feature).
- A CofigProg attribute contains the selection program text that generates a particular configuration when triggered.
- A Configuration feature that is a coherent composition of several features specifying a given product in an SPL. This feature is modeled similar to other elementary features (it can possess a parent feature, versions, attributes, or validity formula, and is controlled by FM constraints) with the exception of a child feature. For example, EuropeConfiguration is a child configuration feature inside a group that contains elementary features (ecall, ERAGlonass) and configuration features (EuropeConfiguration, RussiaConfiguration), as depicted in Fig. 2.

- A Configuration feature management process that manage a configuration feature from its creation to its destruction.

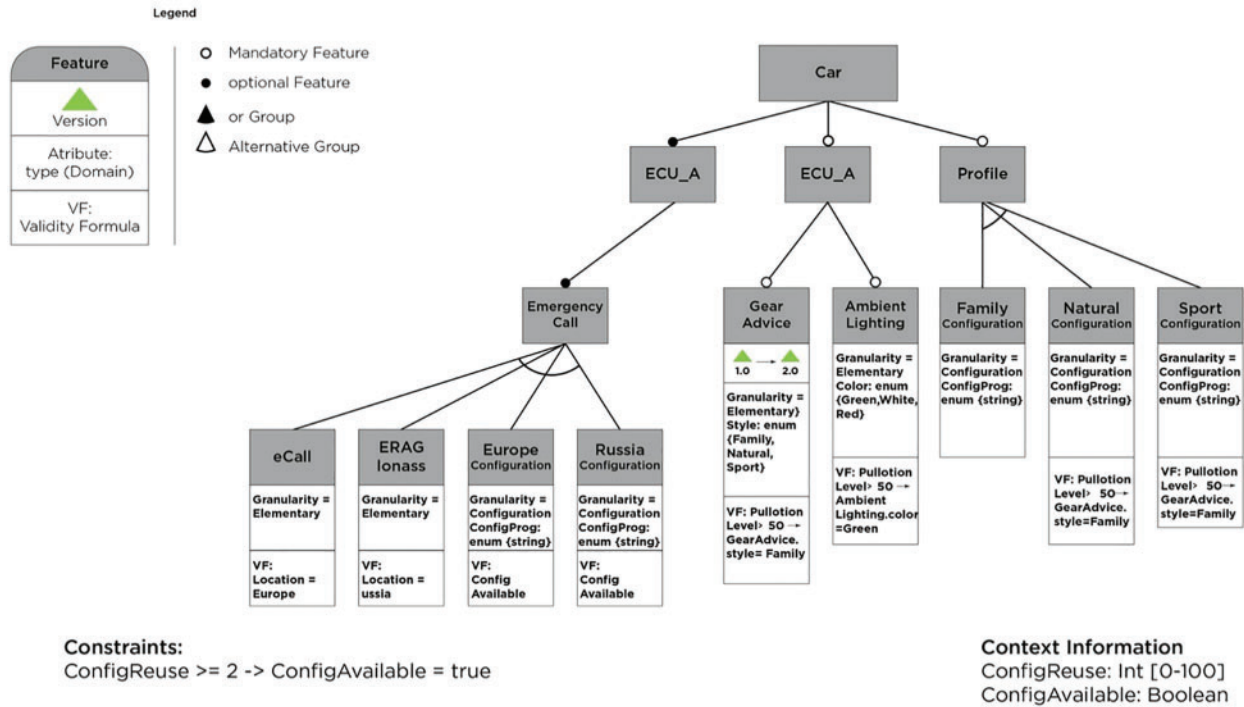


Figure 2: Enhanced feature model for a car system with configuration features

4.3 Configuration Feature Management

A Configuration feature is a coherent composition of a system features along with their relations and constraints to specify a given product in a SPL/DSPL. It is modeled in similar fashion to other elementary features (can be a parent feature or a child feature). It can possess attributes, versions, or validity formula with the exception of possessing a child feature, and it is controlled by FM constraints. For instance, the EuropeConfiguration is a child configuration feature within a group of features that comprises elementary features (ERAGlonass and ecall) and configuration features (Europe Configuration and Russia Configuration), as depicted in Fig. 6.

A configuration feature is modeled by using framework 1 as follows.

Framework 1: Modeling configuration feature

<Configuration feature> → <Configuration feature name>;
 <versions>
 <attribute> = (<attribute name>: <attribute domain>)*
 “VF:” (<validity formula>)*

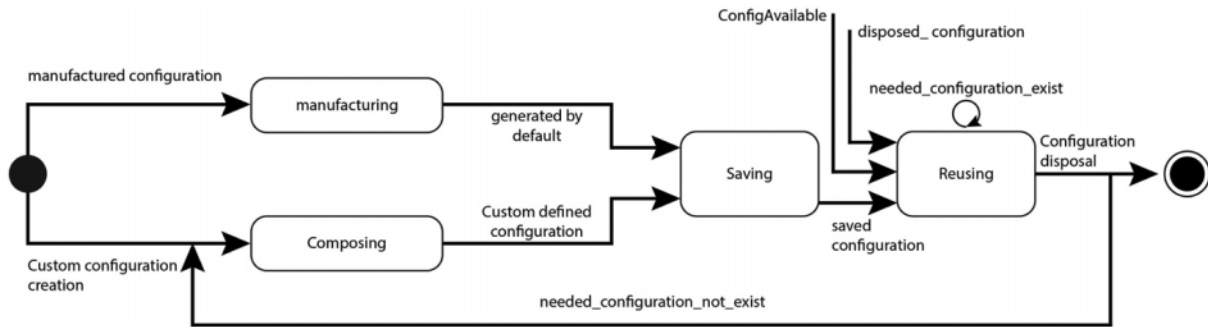


Figure 3: State diagram of the configuration process

4.3.1 Configuration Feature Management Process

The configuration feature management process defines all the states throughout the lifetime of a configuration feature. These states are Manufacturing, Composing, Saving, and Reusing (depicted in Fig. 3). The layout of the configuration process is as follows.

4.3.2 Configuration Process: Life Cycle

→ manufactured configuration

→ configuration creation

manufactured configuration → Default configuration

configuration creation → Composing

generated by default \vee defined configuration → Saving

Saved configuration → Reusing

Reusing \wedge ConfigAvailable \rightarrow needed_configuration_exists \wedge Reusing

Reusing \wedge ConfigAvailable \rightarrow needed_configuration_does_not_exist \wedge Composing

Reusing \wedge disposed configuration \rightarrow exit

Two types of configuration features are supported — a default configuration feature that is generated by the manufacturer and a custom defined configuration feature that is explicitly composed by a customer or the system depending on a particular context. The HyVarRec [2] reconfiguration engine is responsible for creating and validating these configurations and the created configuration is saved within the assets database to be reused every time it fulfills the requirements of a new context. The states of the two configuration features (depicted in Fig. 3) are as follows.

- Composing State: composing state is specified in framework 2 as follows.

Framework 2: Modeling composing state

composing \rightarrow ["Defined configuration" <configuration name>] \wedge
 ["ConfigProg" {"enabled features"
 (<enable feature>)* \vee "disabled features" (<disable feature>)*}]
 + ["VF:"(<Formula>)*]

- Saving State: saving state is specified in framework 3 as follows.

Framework 3: Modeling saving state

Saving \rightarrow update <FM> \wedge update Assets

update <FM> \rightarrow insert_feature<configuration name>: [<version>*,<attribute>*,<VF>*]

update asset \rightarrow save_in_DB<configuration name>

- Reusing State: The state, IndirectReuse, corresponds to the state of a configuration feature that might be reused as a component of another configuration feature (integration of a configuration feature with another to serve a specific context). For instance, the preference configurations for driving style (sports, natural, family, and emergency) may be saved using this state as the emergency configuration might reuse any of the sports, natural, or family configurations in order to adapt to the context of the emergency. This type of reuse is introduced alongside the imply and exclude constraints.

Alternatively, a configuration feature might be reused directly, corresponding to the DirectReuse. For instance, triggering the reconfiguration process each time the car changes its location from Europe to Russia and vice versa, which is a frequent event, is a DirectReuse feature. In such a case, keeping the appropriate configuration feature available to reuse each time the frequent event occurs decreases the effort and time required by the reconfiguration process. The reusing state is specified in framework 4 as follows.

Framework 4: Modeling reusing state

Reusing \wedge ConfigAvailable \rightarrow search_configuration(configuration name) \wedge
 needed_configuration_exist \wedge
 configuration_reuse(configuration name)

Reusing \wedge ConfigAvailable \rightarrow search_configuration(configuration name) \wedge
 needed_configuration_not_exist \wedge Composing

Reusing \wedge disposal_configuration \rightarrow Exit

4.4 Configuration Management-Examples

4.4.1 Creation of the EuropeConfiguration Feature

The text program of the composing, saving, and reusing states are shown by the following example (depicted in Fig. 4).

4.4.2 Creation of the Sporty Driver Profile Feature

The text program of the composing, saving, and reusing states are shown by the following example (depicted in Fig. 5).

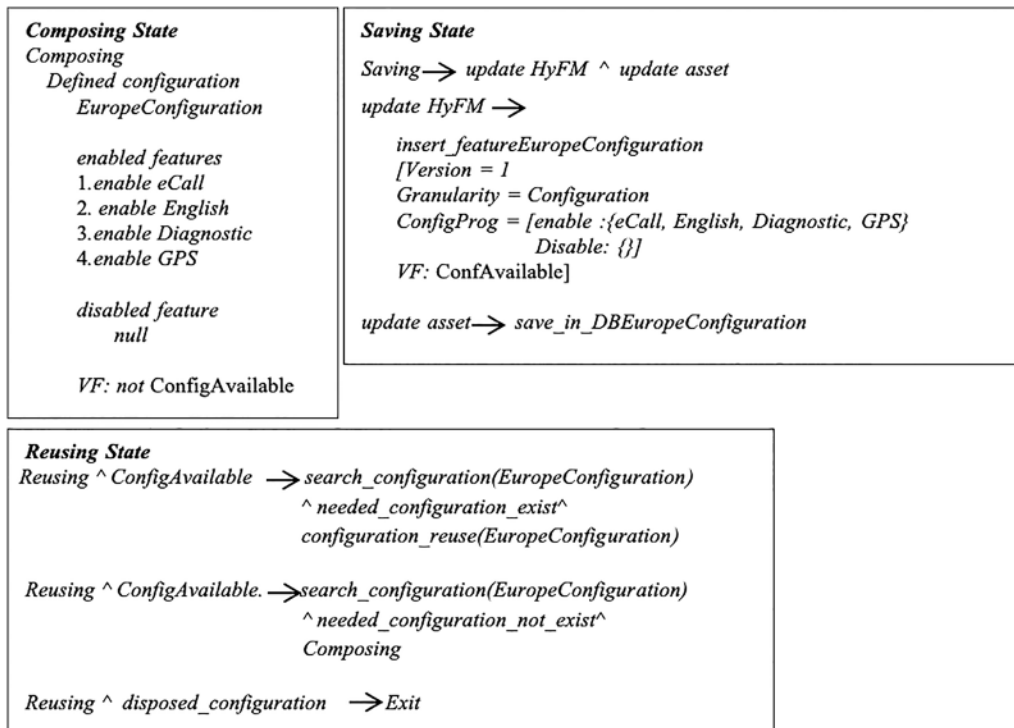


Figure 4: Creation of the EuropeConfiguration feature

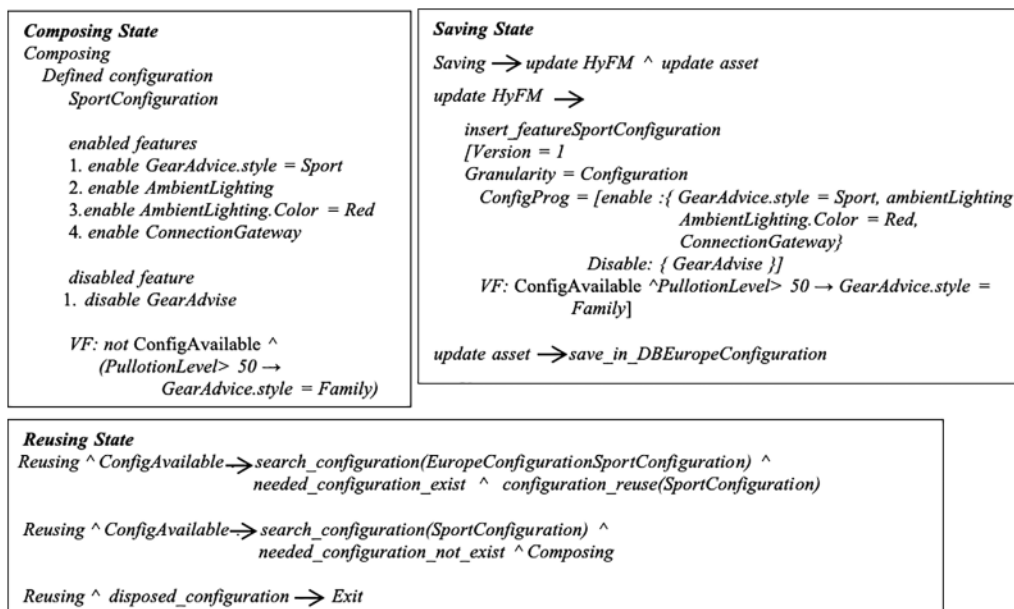


Figure 5: Creation of the sporty driver profile feature

5 Case Study

The tool, DarwinSPL [31], is used to study the feasibility of the illustrative example (discussed in Sections 3 and 4.3). First, the three driving styles of the car system—family, natural, and sports—are generated by the manufacturer as configuration features (FamilyConfiguration, NaturalConfiguration, and SportConfiguration) under the control of the configuration feature management process. They are designated to be group members with profile as the parent feature, as demonstrated in Section 4.3. Further, the three configuration features are saved within the assets database. They remain available for reuse depending on the context. Fig. 6 depicts a screenshot of the FM following the creation of the three driving styles, modeled using the FM editor of DarwinSPL.

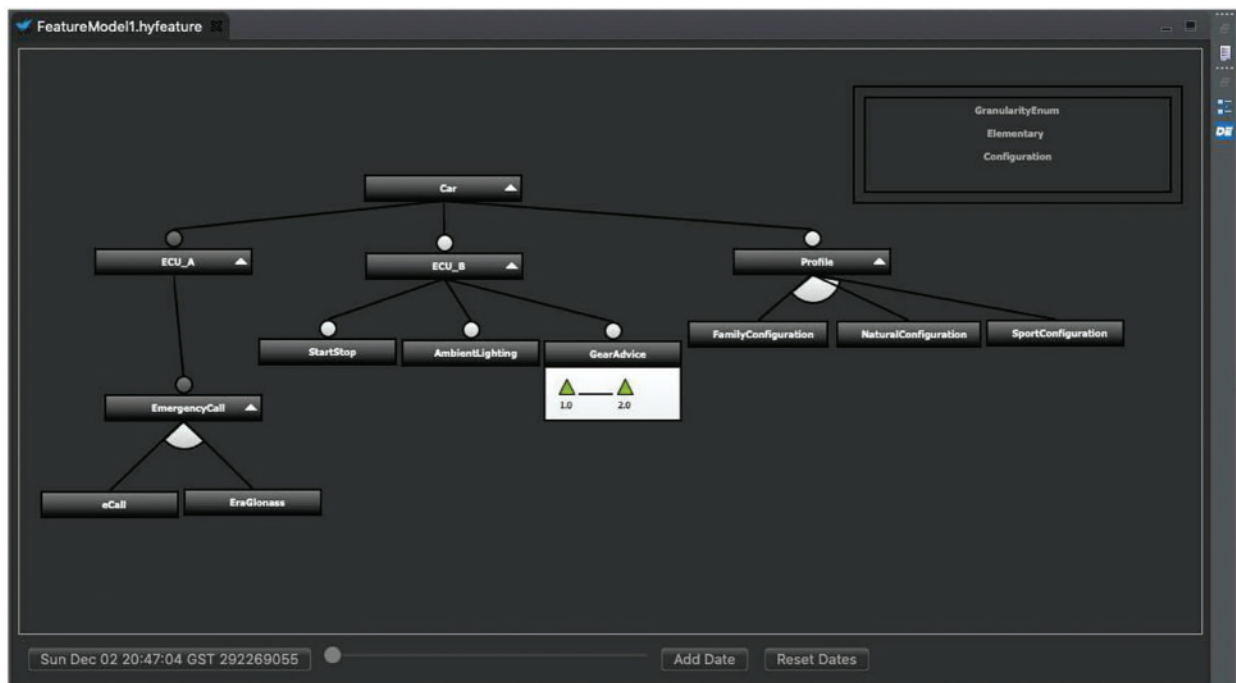


Figure 6: Screenshot of the FM, modeled using the FM editor of DarwinSPL

Firstly, assuming that the driver selects the SportConfiguration feature as his/her preferred configuration, this configuration will be reused as long as it fulfills the requirements of the current context. For instance, if Pollution Level > 50, then the SportConfiguration feature does not remain operational, as it does not fulfill the requirements of the current context. Hence, the HyVarRec reconfiguration engine will replace it with the FamilyConfiguration feature, as determined by the VF and CTC of the FM. Moreover, another person driving this car might change the preferred configuration to either the NaturalConfiguration feature or FamilyConfiguration feature based on his/her preferences.

Secondly, assuming that the driver drives frequently from Europe to Russia, the reconfiguration engine will be triggered each time he/she changes his/her location in order to provide a suitable configuration for the current context [2]. For instance, when the driver is in Europe, the reconfiguration process is triggered to compose a configuration for this context (Location=Europe)

and the value of the context element sensor ConfigReuse is set to 1, as outlined in Section 4.3. Now, if the driver drives to Russia and back to Europe, the reconfiguration process is triggered again to compose a configuration for the same context (Location=Europe) and the value of the sensor, ConfigReuse, is set to 2. This prompts the configuration feature management process to save this configuration as the EuropeConfiguration feature within the assets database, and the value of the sensor ConfigAvailable is set to true. Subsequently, the FM is updated using DarwinSPL. Fig. 7 depicts the updated FM with the configuration features, modeled using the FM editor of DarwinSPL. Figs. 8–10 illustrate the modeling of the newly added contextual elements, validity formula, and constraints, as modeled using DarwinSPL.

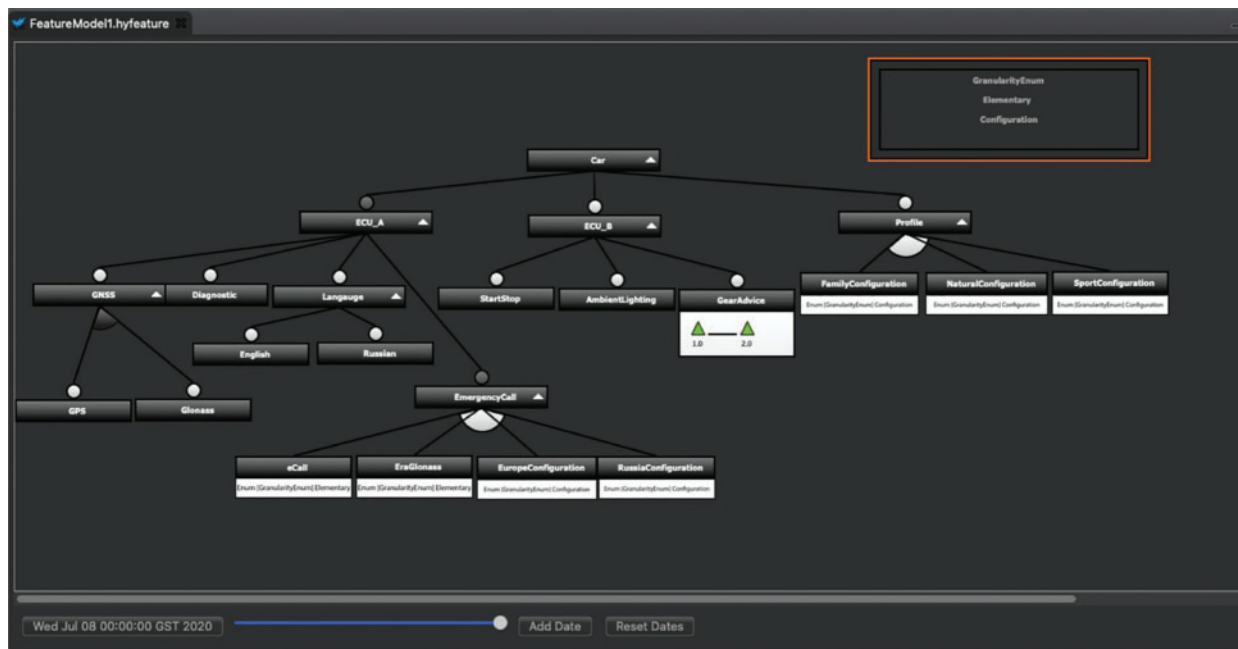


Figure 7: Updated FM with the configuration features, using the FM editor of DarwinSPL

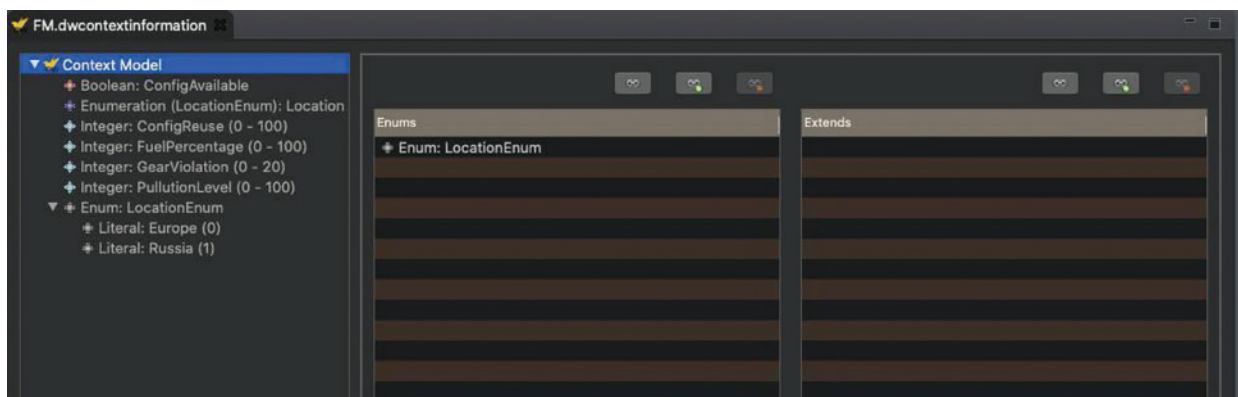
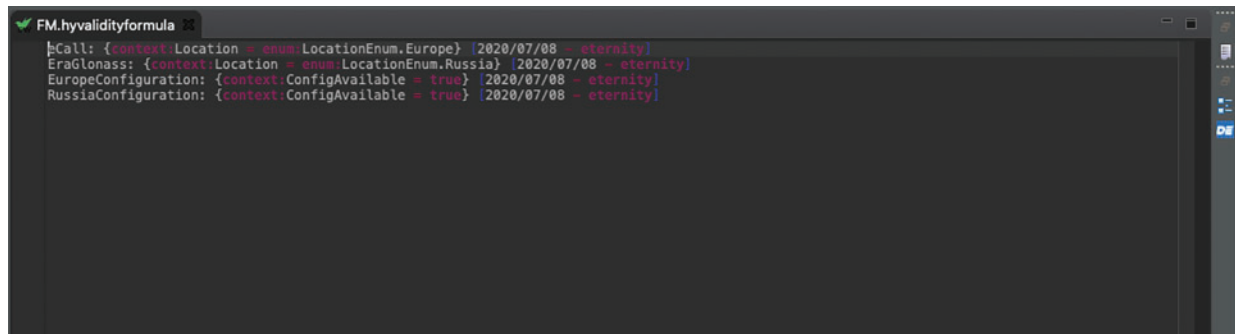


Figure 8: Contextual elements of the FM, using the contextual information editor of DarwinSPL

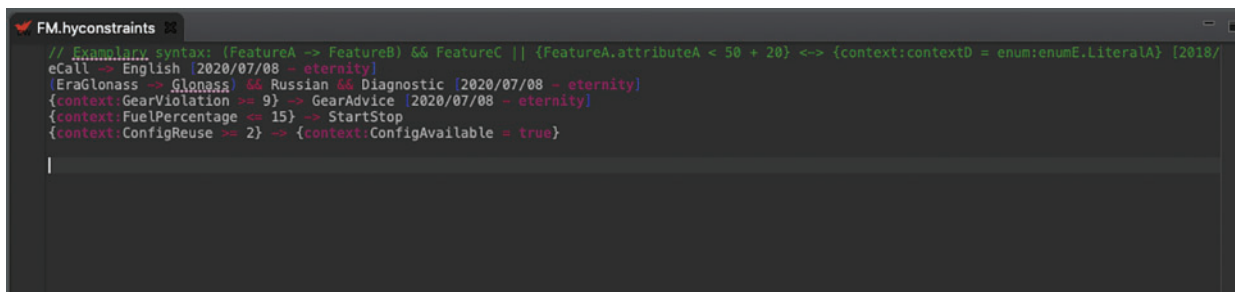


```

FM.hyvalidityformula
[Call: {context:Location = enum:LocationEnum.Europe} [2020/07/08 - eternity]
EraGlonass: {context:Location = enum:LocationEnum.Russia} [2020/07/08 - eternity]
EuropeConfiguration: {context:ConfigAvailable = true} [2020/07/08 - eternity]
RussiaConfiguration: {context:ConfigAvailable = true} [2020/07/08 - eternity]

```

Figure 9: VF of the FM, modeled using the VF editor of DarwinSPL



```

FM.hyconstraints
// Exemplary syntax: (FeatureA -> FeatureB) && FeatureC || {FeatureA.attributeA < 50 + 20} <-> {context:contextD = enum:enumE.LiteralA} [2018/
eCall -> English [2020/07/08 - eternity]
EraGlonass -> Glonass && Russian && Diagnostic [2020/07/08 - eternity]
{context:GearViolation >= 9} -> GearAdvice [2020/07/08 - eternity]
{context:FuelPercentage <= 15} -> StartStop
{context:ConfigReuse >= 2} -> {context:ConfigAvailable = true}

```

Figure 10: Constraints of the FM, modeled using the constraints editor of DarwinSPL

This configuration feature is reused based on the context until the context changes; e.g., the driver drives from Europe to Russia. When this happens, other configuration features of the system are reused depending on the context until they are to be disposed. This procedure enables the obtainment of reliable results by expending considerably less effort over considerably shorter durations of time, as the reconfiguration process does not need to be triggered as many times due to the utilization of the reuse process.

6 Evaluation

Based on current research in the domain of feature modeling techniques, SASs and reuse processes, the following criteria are selected to evaluate the proposed scheme and compare its performance with those of existing methods.

- (1) Development of SASs using reuse process support.
- (2) Supporting the configuration feature concept using FMs.
- (3) Efficient reuse of configuration features.
- (4) Managing product customization during run time.

In a study [2], HyFM was introduced, which supports feature attributes, contextual information, and relations between contextual information and features selected based on current context and constrained by VFs, cross-tree constraints, a finite set of feature versions, version-aware constraints, and cardinality. These additional concepts enable HyFM to model and manage SASs by focusing on the representation and reuse of elementary features without considering the modeling or reuse of configuration features. As the proposed system is integrated with HyFM,

it is capable of supporting SAS modeling and management, as well as modeling and reusing configuration features. Thus, the proposed method satisfies all of the aforementioned criteria.

Whereas in [21], the authors presented an FM as a two-layer model consisting of application and infrastructure layers to manage the required configuration that satisfy the stakeholders' desires and the potential of the platform of available implementations. In addition, this FM is capable of defining two types of constraints—inner constraints among features belonging to the same layer and intra-constraints among features belonging to different layers. However, this FM manages and reuses only elementary features, without considering the management and reuse of configuration features. Further, it does not support product customization during run time.

In another study [26], the FM was extended using binding time constraints to restrict SAS-staged reconfiguration processes. The concepts of configuration stages, used to define arrangements among features and differentiate between dynamic and static binding times of features; complex binding time constraints, used to restrict product configuration processes by taking logical consistency and arrangement restrictions of feature selections into consideration; and stateful reconfiguration models, used to control possible sequences of configuration changes of dynamic features during the product's life cycle were introduced in the indicated paper. However, these configurations were not modeled using FMs.

Whereas an autonomic software product line engineering (ASPL) methodology was presented in another study [32]. This methodology allows developers to develop SASs with reuse processes by selecting, specializing, and subsequently, integrating reusable artifacts. This methodology used FMs to model only the scope definition of ASPL; hence, no concepts were further added to the FM to support the modeling of the developed SAS with reuse.

The conclusions of the aforementioned comparative analysis have been depicted in [Tab. 1](#), where the criteria used have been numbered sequentially from 1 to 4.

Table 1: Comparison between various configuration management schemes modeled using FM

Criterion	[2]	[21]	[26]	[32]	[this works]
1	✗	✗	✗	✓	✓
2	✗	✗	✗	✗	✓
3	✗	✗	✗	✗	✓
4	✓	✗	✗	✗	✓

7 Conclusion

In one of the most significant tasks in the domain of FM-based SASs [2], the authors introduced a framework for SPLs that allow modeling of contextual influences, user customization, and evolution. In particular, a hybrid feature model (HyFM) was presented. It supports contextual information, validity formulae, cross-tree constraints, and version awareness. However, this approach is incapable of managing configuration reuse. Moreover, a reconfiguration process is initiated every time the context changes or an artifact of the system is evolved by an engineer. This escalates the required time and resources. To rectify this shortcoming, an extension to HyFM [2] was proposed in this study to manage configuration features that support its modeling and coherent reuse process. This extension includes new features (configurations), context information

and constraints to ensure coherency among features. Configuration feature management process has been defined to manage configuration features and their reuse process from creation to destruction. The fundamental complexity of the management of configuration reuse processes lies in the adaptation of reused configuration features. Adaptation before reuse becomes necessary, as the configuration to be reused might no longer be suitable for the new context. Further, once a configuration feature is created, it requires continuous maintenance in the repository, new configuration features need to be saved, configuration features with errors need to be tracked, and configuration features that are no longer being reused for any reason need to be removed. The complexity of configuration reuse process management, the variability of configuration features, their optimal selection, and their evolution remain major challenges that may be addressed in future works.

Acknowledgement: This work is completely supported by Philadelphia University through the research project: Bio-Inspired Self-Adaptive Systems Variability Modelling, RP 12/2017.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

- [1] M. Carvalho, M. Da Silva, G. S. Gomes, A. R. Santos and I. Machado *et al.*, “On the implementation of dynamic software product lines: An exploratory study,” *The Journal of Systems and Software*, vol. 136, pp. 74–100, 2018.
- [2] J. Mauro, M. Nieked, C. Seidl and I. Chieh-yu, “Context-aware reconfiguration in evolving software product lines,” *Science of Computer Programming*, vol. 163, pp. 139–159, 2018.
- [3] N. Padhy, R. P. Singh and S. C. Satapathy, “Software reusability metrics estimation: Algorithms, models and optimization techniques,” *Computer Electronic Engineering*, vol. 69, pp. 653–668, 2018.
- [4] R. Capilla, B. Gallina, C. Cetina and J. Favaro, “Opportunities for software reuse in an uncertain world: From past to emerging trends,” *Journal of Software: Evolution and Process*, vol. 8, pp. e2217, 2020.
- [5] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl and P. Steenkiste, “Rainbow: Architecture-based self-adaptation with reusable infrastructure,” *Computer*, vol. 10, pp. 46–54, 2004.
- [6] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen and K. Lund *et al.*, “Using architecture models for runtime adaptability,” *IEEE Software*, vol. 2, pp. 62–70, 2006.
- [7] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen and S. Hallsteinsen *et al.*, “Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments,” *In Software Engineering for Self-Adaptive Systems*, Springer, Berlin, Heidelberg, pp. 164–182, 2009.
- [8] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar and I. Habli *et al.*, “Engineering trustworthy self-adaptive software with dynamic assurance cases,” *IEEE Transactions on Software Engineering*, vol. 11, pp. 1039–1069, 2018.
- [9] M. Marques, J. Simmonds, P. O. Rossel and M. C. Bastarrica, “Software product line evolution: A systematic literature review,” *Information and Software Technology*, vol. 105, pp. 190–208, 2019.
- [10] A. Tahir, S. A. Licorish and S. G. Macdonell, “Feature evolution and reuse-an exploratory study of eclipse,” in *2017 24th Asia-Pacific Software Engineering Conf. (APSEC)*, Nanjing, China, pp. 582–587, 2017.
- [11] E. N. Teixeira, F. A. Aleixo, F. D. Amâncio, J. E. Oliveira and U. Kulesza *et al.*, “Software process line as an approach to support software process reuse: A systematic literature review,” *Information and Software Technology*, vol. 116, no. 1, pp. 106–175, 2019.

- [12] M. B. Duran and G. Mussbacher, "Reusability in goal modeling: A systematic literature review," *Information and Software Technology*, vol. 110, pp. 156–173, 2019.
- [13] M. Naeem, "Matching of service feature diagrams based on linear logic," Ph.D. dissertation, University of Leicester, UK, 2012.
- [14] G. Assad, M. Naeem and H. Abdulwahab, "Towards cardinality-based service feature diagrams," *Computational Ecology and Software*, vol. 5, no. 1, pp. 69–76, 2015.
- [15] O. Younes and S. Ghoul, "Systems versioning: A features-based meta-modeling approach," *International Science Index*, vol. 8, no. 6, pp. 988–992, 2014.
- [16] P. Arcaini, A. Gargantini and M. Radavelli, "Achieving change requirements of feature models by an evolutionary approach," *The Journal of Systems and Software*, vol. 150, pp. 64–76, 2019.
- [17] M. Nieke, C. Seidl and S. Schuster, "Guaranteeing configuration validity in evolving software product lines," in *Proc. VaMoS '16, ACM*, New York, NY, USA, pp. 73–80, 2016.
- [18] T. Clark, F. Ulrich, I. Reinhartz-berger and A. Sturm, "A multi-level approach for supporting configurations: A new perspective on software product line engineering," in *Proc. ER Forum*, Valencia, Spain, pp. 170–178, 2017.
- [19] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou and G. Saake, "Personalized recommender systems for product-line configuration processes," *Computer Languages, Systems & Structures*, vol. 54, pp. 451–471, 2018.
- [20] Y. Li, T. Yue, S. Ali and L. Zhang, "Enabling automated requirements reuse and configuration," *Software & Systems Modeling*, vol. 18, no. 3, pp. 2177–2211, 2019.
- [21] E. D. Farahani and J. Habibi, "Feature model configuration based on two-layer modeling in software product lines," *International Journal of Electrical & Computer Engineering*, vol. 9, no. 4, pp. 2088–2108, 2019.
- [22] B. I. Moritani and J. Lee, "An approach for managing a distributed feature model to evolve self-adaptive dynamic software product lines," in *21st Int. System and Software Product Line Conf. (SPLC)*, Sevilla, Spain, pp. 107–110, 2017.
- [23] M. Bashari, E. Bagheri and W. Du, "Self-adaptation of service compositions through product line reconfiguration," *Journal of Systems and Software*, vol. 144, pp. 84–105, 2018.
- [24] C. h. Werner, S. Werner, R. Schone, S. Gotz and U. Aßmann, "Self-adaptive synchronous localization and mapping using runtime feature models," in *The 7th Int. Conf. on Data Science, Technology and Applications*, Porto, Portugal, pp. 409–418, 2018.
- [25] A. Mousa, J. Bentahar and O. Alam, "Context-aware composite saas using feature model," *Future Generation Computer Systems*, vol. 99, pp. 376–390, 2019.
- [26] M. Lochau, J. Bürdek, S. Hölzle and A. Schürr, "Specification and automated validation of staged reconfiguration processes for dynamic software product lines," *Software & System Modeling*, vol. 16, no. 1, pp. 125–152, 2017.
- [27] S. Ibraheem and S. Ghoul, "Software evolution: A features variability modeling approach," *Journal of Software Engineering*, vol. 11, no. 1, pp. 12–21, 2017.
- [28] D. Benavides, P. Trinidad and A. Ruiz-corteés, "Automated reasoning on feature models," *Advanced information systems engineering, CAiSE, lecture notes in computer science*, Berlin Heidelberg, vol. 3520, pp. 491–503, 2005.
- [29] C. Seidl, I. Schaefer and U. Aßmann, "Capturing variability in space and time with hyper feature models," in *Proc. VaMoS '14*, Nice, France, pp. 6–7, 2014.
- [30] M. Nieke, J. Mauro, C. Seidl and I. C. Yu, "User profiles for context-aware reconfiguration in software product lines" in *Int. Symposium on Leveraging Applications of Formal Methods*, Imperial, Corfu, Greece, pp. 563–578, 2016.
- [31] M. Nieke, G. Engel and C. Seidl, "DarwinSPL: An integrated tool suite for modeling evolving context-aware software product lines," in *Proc. VAMOS '17*, New York, NY, USA, pp. 92–99, 2017.
- [32] N. Abbas, J. Andersson and D. Weyns, "ASPL: A methodology to develop self-adaptive software systems with systematic reuse," *The Journal of Systems and Software*, vol. 167, pp. 1–19, 2020.